New Java Language Features coming in JDK1.5

Simon Ritter
Technology Evangelist
simon.ritter@sun.com

Java Language Changes

- JDK 1.0
  - Initial language, very popular
- JDK1.1
  - Inner classes, new event model
- JDK 1.4
  - Assertions (minor change)
- JDK 1.5
  - Biggest changes to language since release

Java 2 Standard Edition 1.5

- Seven major new features
  - Generics
  - Enhanced for loop (“foreach”)
  - Autoboxing/Unboxing
  - Type-safe enumerations
  - Varargs
  - Static import
  - Metadata
- ALL PENDING JSR EXPERT GROUP CONFIRMATION

Generics

- Problem: Collection element types
  - Cannot be checked at compile time
  - Assignment must use cast
  - Can cause runtime errors (`ClassCastException`)
- Solution:
  - Tell the compiler what type your collection is
  - Compiler can fill in casts for you
  - Guaranteed to succeed *

* As long as all code is generic

Generics Example

- Old code

```
List l = new LinkedList();
l.add(new Integer(0));
Integer i = (Integer)l.iterator.next();
```

- New code

```
List<Integer> l = new LinkedList<Integer>();
l.add(new Integer(0));
Integer i = l.iterator.next();
```

Generics

- Generics are NOT templates
  - No code size increase
  - No hideous complexity
  - No “template metaprogramming”

Enhanced for Loop (foreach)

- Problem:
  - Iterating over collections is tricky
  - Often, iterator only used to get an element
  - Iterator is error prone (Occurs three times in a for loop)
  - Can produce subtle runtime errors
- Solution: Let the compiler do it
  - New for loop syntax

```
for (variable : collection)
```

Enhanced for loop example

- Old code

```
void cancelAll(Collection c) {
  for (Iterator i = c.iterator(); i.hasNext(); ) {
    TimerTask task = (TimerTask)i.next();
    task.cancel(); } }
```

- New Code

```
void cancelAll(Collection<TimerTask> c) {
  for (TimerTask task : c)
    task.cancel();
}
```

- Also works for arrays

Auto-boxing of primitive types

- Problem:
  - Conversion between primitive types and wrapper objects (and vice-versa)
  - Needed when adding primitives to a collection
- Solution: Let the compiler do it

```
Integer intObj = 22;    // Boxing conversion
int i = (int)intObj    // Unboxing conversion


ArrayList al = new ArrayList();
al.add(22);  //  Boxing conversion
```

Typesafe Enumerations

- Problem:
  - Variable needs to hold limited set of values
    - e.g. Card suit can only be Spade, Diamond, Club, Heart
- Solution: New type of class declaration
  - enum type has public, self-typed members for each enum constant
  - new keyword, enum
  - works with switch statement

Enumeration Example


```
public enum Suit { spade, diamond, club, heart } ;
public enum Value { ace, two, three, four, five,
                    six, seven, eight, nine, ten,
                    jack, queen, king } ;

List<card> deck = new ArrayList<card>();

for (Suit suit : Suit.VALUES)
    for (Value value : Value.VALUES)
      deck.add(new Card(suit, value);

Collections.shuffle(deck);
```

Think how much JDK1.4 code this would require!

Enumeration Examples


```
public enum Coin {
  penny(1), nickel(5), dime(10), quarter(25);

  Coin(int value) { this.value = value; }
  private final int value;
  public int value() { return value; }
}
```

Varargs

- Problem:
  - To have a method that takes a variable number of parameters
  - Can be done with an array, but not nice
  - Look at **`java.text.MessageFormat`**
- Solution: Let the compiler do it for you
  - New syntax:

```
public static String format(String fmt,
    Object... args);
```

  - Java gets **`printf`** !!!

Static imports

- Problem:
  - Having to fully qualify every static referenced from external classes
- Solution: New import syntax
  - import static TypeName . Identifier ;
  - import static Typename . * ;
  - Also works for static methods and enums

    e.g **`Math.sin(x)`** becomes **`sin(x)`**

Metadata

- Problem:
  - Some APIs require lots of standard code
  - How to indicate this to a tool
- Solution: Annotated source code
  - e.g.

```
@remote getPrice(Product p)
```

Concurrency Utilities

- Goal: Beat C performance in high end server side applications
- New framework for locks to provide greater flexibility over synchronized
- No more threads, use Executors
  - Use anExecutor.execute(aRunnable)
  - Not new Thread(aRunnable).start();
- Runnable and Callable
  - Callable for things that return values and/or exceptions

Performance Improvements

- Goals
  - 25% improvement in startup time over 1.4.0
  - Asynchronous I/O
  - Unsynchronized StringBuffer class
  - Non-blocking equivalents of SSLSocket and SSLServerSocket
  - Reduce memory footprint of JVM
  - Improve classloader speed
  - Add concurrency library to Java core

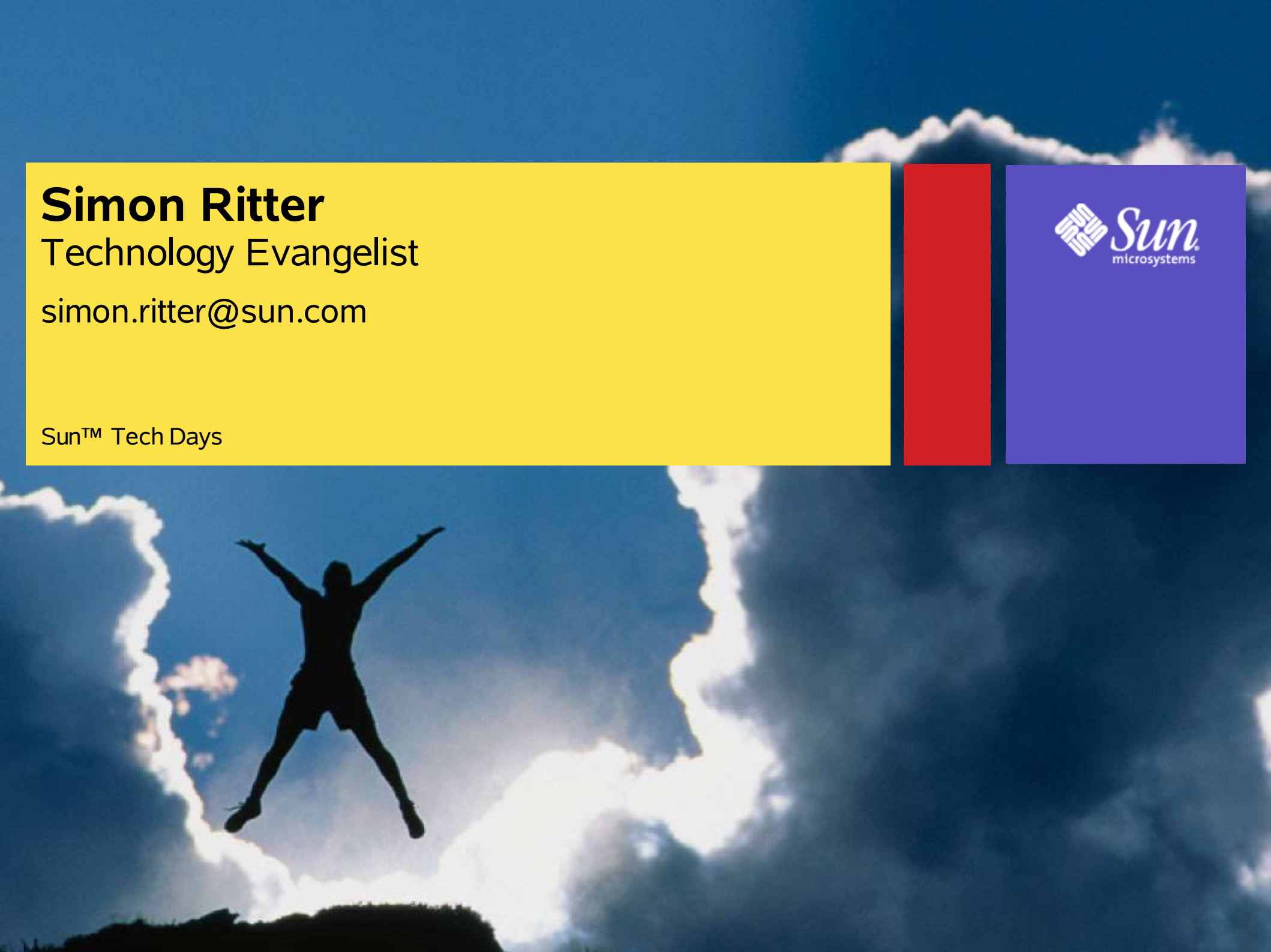
Remember!

All features of JDK1.5 just described are subject to change before release.

Release provisionally planned for Summer 2004, beta version at end of 2003

Further information

- www.jcp.org
  - JSR-059 J2SE 1.4 Feature definition
  - JSR-014 Generics
  - JSR-133 Revised memory model
  - JSR-166 Concurrency utilities
  - JSR-175 Metadata facility
  - JSR-201 Enums, Autoboxing, For loop, Static import
- java.sun.com/j2se



Simon Ritter

Technology Evangelist

simon.ritter@sun.com

Sun™ Tech Days